:: IJEEIT ::
(International Journal of Electrical Engineering and Information Technology)
Volume 03 Number 02 September 2020

ISSN : 2615-2096 (ONLINE)
ISSN : 2615-2088 (PRINTED)

Implementation of Docker and Continuous Integration / Continuous Delivery for Management Information System Development

Akbar Dhany Widiyanto
Faculty of Computer Science, Narotama University, Indonesia
edu.akbardhany@gmail.com

Benediktus Anindito
Faculty of Computer Science, Narotama University, Indonesia
benediktus.anindito@narotama.ac.id

Moh. Noor Al Azam
Faculty of Computer Science, Narotama University, Indonesia
noor.azam@narotama.ac.id

ABSTRACT

A series of Development and Operations (DevOps) in the process of making the Narotama University Management Information System have not been implemented properly by previous developers. There are improvements or additional features of the Management Information System that are in accordance with the functionality and the increasing development needs that will be used by the academic community, so that the Management Information System developer has a little difficulty in integrating documents and distributing applications with different packages to the Production Server. In this study, a new system design is proposed by applying the practice of Continuous Integration / Continuous Delivery as a document integration process and can simplify the application distribution process, as well as implementing the Docker Container Platform as an application container with different packages that can be run on production server together. The results of implementing the practice of Continuous Integration / Continuous Delivery and the implementation of the Docker Container Platform are able to help integrate documents between developers and be able to release fixes and add features packaged in different containers automatically and periodically without long delays. which only takes an average of 17.9 seconds in the process of sending the application to the Production Server.

Keywords: *Continuous Delivery, Continuous Integration, Docker*

INTRODUCTION

The development of computer technology utilization through the internet network, better known as Cloud Computing, is currently growing rapidly, especially in Indonesia. Moreover, Cloud Computing technology is indispensable in the process of creating and developing an application system between developers. In Indonesia, 19.4% of companies have started using public cloud services, while 32.1% of companies say they will use cloud services within the next year (Threestayanti, 2019). At the stage of application creation and development, Development and Operations (DevOps) is a series of implementations to connect between developers automatically over an internet network that is useful for faster and betterly organized build, test, and release processes. Global research conducted by CA Technologies recently revealed that 57% of companies in Indonesia have achieved advanced DevOps Maturity (Advanced DevOps Maturity), a much higher percentage than other countries located in the Asia Pacific region and Japan which average 38% (Admin i3, 2019). Some practices in DevOps include Continuous Integration (CI), Continuous Delivery (CD) (Erich, 2017), and Build: Container Platform. The implementation of DevOps in its entirety and only applied part of some of the practices in it, still passively carried out on the development of information management application system among academic

ISSN : 2615-2096 (ONLINE)
ISSN : 2615-2088 (PRINTED)

education. This set of CI and CD practices in DevOps has advantages, some of which are that CI practices can help with the regular release of more application projects, and CD practices can deliver application packages on the production server as soon as possible. However, in the process of developing Management Information System (SIM) Narotama University is still not implementing CI / CD practice in the framework of DevOps. When the developer will update the release for the SIM, it will take a long time to do so. In the above issue, there is a system design that can minimize the work, namely using a series of CI / CD practices in DevOps. With the acceleration of devops development today, the use of CI/CD practices has been implemented in several previous studies. Research conducted by Arachichi, et al (Arachchi, 2018) explains that the implementation of CI/CD practices successfully makes the delivery process efficient and can increase system productivity. Shanin's research, et al (Shahin, 2017) explains that Continuous Practices has been successfully applied to projects created from scratch as well as maintenance of existing projects. In this study, continuous integration / continuous delivery (CI / CD) workflow practices were designed for the process of integrating git repositories into production servers that benefited the delivery process into efficiency and increased system productivity,(Arachchi, 2018) and implemented into server virtualization techniques using Docker for the purpose of facilitating the release of the application development.

RESEARCH METHODS

The design of the system design that will be applied to this research includes inputs, processes, and outputs. The design of the system design will be spelled out in Figure 1.

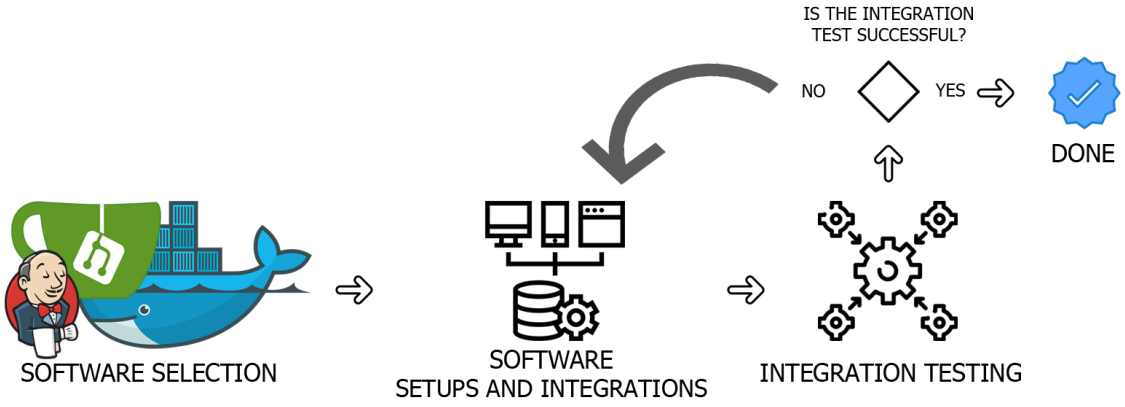

Figure 1. Design System Plan

1. Software Selection

The selection of software is based on the initial purpose of this research, the software is Gitea, is an open-source based Git Server repository manager (Gitea, 2020) that serves as a folder file in project; Jenkins, is an open-source automation server that can help automatically execute software development commands that include build, test, and deploy (Jenkins Developers, 2018) to facilitate Continuous Integration / Continuous Delivery (CI / CD); Docker, is an open-platform for developers and systems to develop, send, and run applications and can separate between infrastructures for those applications (Docker Documentation, 2020).

2. Software Setups and Integrations

The stages of software setup and integration implemented in this research will be divided into 2 (two) sections namely the Software Setup Stage and the Software Integration Stage which will be spelled out in Figure 2.

ISSN : 2615-2096 (ONLINE)
ISSN : 2615-2088 (PRINTED)

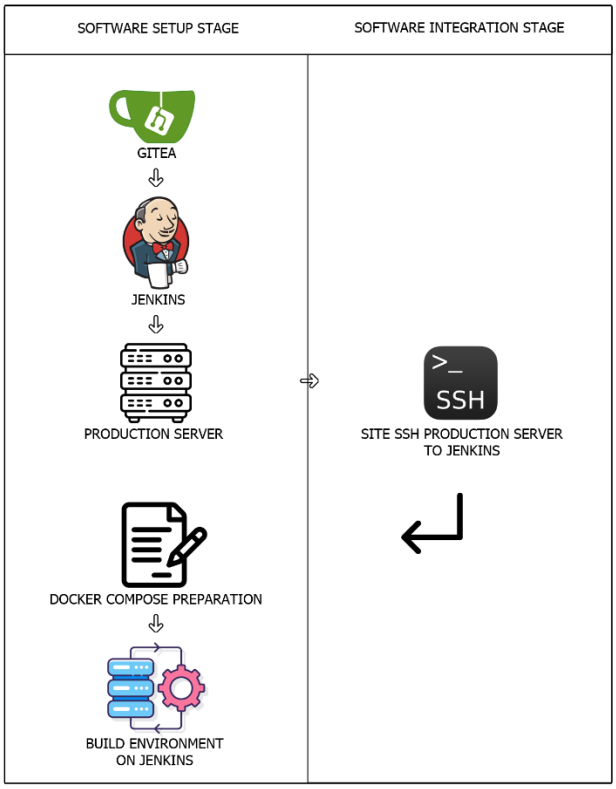

Figure 2. Software Setup Stage and Software Integration Stage

2.1. Gitea

In the first stage of Software Setup is to install Gitea (Repository Manager) application on the Server, which serves as the Main Repository for a project to be created collectively between developers.

2.2. Jenkins

Next in the second stage of Software Setup is installing the Jenkins application on the Server, which serves as a server automation to connect Gitea (Repository Manager) with Production Server through a pre-prepared Job. Then install Secure Shell (SSH) on Jenkins Plugins. SSH is a method of securing remote logins from 1 (one) computer to another for strong authentication, security protection, and strong encryption communication (Ylonen, 1996).

2.3. Production Server

The third stage of Software Setup is to configure the SSH server through generate SSH Key in preparation for Continuous Integration / Continuous Delivery (CI / CD). Next, set up the folder in preparation for the container for the CI / CD.

2.4. Site SSH Production Server to Jenkins

For the Software Integration Stage, the first creates credentials on Jenkins for permissions that serve as authentication from Production Server. And then set up SSH Remote Hosts as the communication address on the Production Server, Gitea Servers in the Jenkins system configuration as well as create Bash-based file (Unix Shell) containing commands for repository updates, Docker Image builds, and Docker Compose execution.

2.5. Docker Compose Preparation

Docker Compose is specifically designed to facilitate interaction in multiple Docker Container (List, 2017). Docker Compose is written on the .YAML (Yet Another Markup Language) aims to define what needs to be used in the Docker Container to be created and executed in unison.

ISSN : 2615-2096 (ONLINE)
ISSN : 2615-2088 (PRINTED)

2.6. Build Environment on Jenkins

In the last stage of the Software Setup section, the Build Environment used on Jenkins is execute Shell Script on Remote Host Using SSH which populates site SSH and sets the command to run bash files located on Production Server, as well as set Build Periodically to run Job on Jenkins automatically and periodically.

3. Integration Testing

In the design section of the last system design is the testing of the results of the Software Setup and Integration process above. Integration testing between Production Server and Jenkins through SSH Remote Hosts has been successfully run as shown in Figure 3.

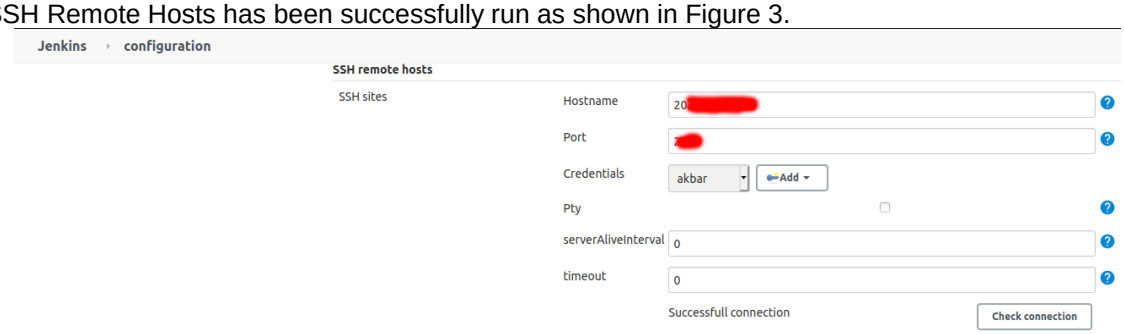

Figure 3. Integration Testing

RESULT AND DISCUSSIONS

The result of this research is the establishment of a new system design for the implementation of Docker and Continuous Integration / Continuous Delivery (CI / CD) implementations on Production Server as described in Figure 4.

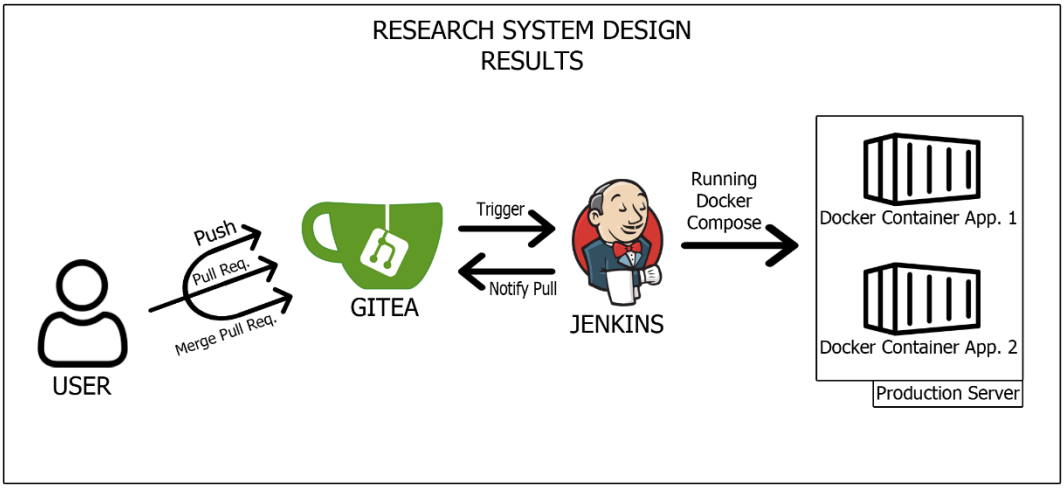

Figure 4. Research System Design Results

The implementation of Continuous Integration / Continuous Delivery (CI / CD) and Build: Container Platform practices in this study requires interconnected credentials between Gitea acting as Repository Manager, Jenkins acting as Server Automation, and Production Server acting as the deployment place of the application. The use of Docker Container in this research is able to run multiple applications well simultaneously on the Production Server.

While the procedure for running bash files executed on Jenkins Job is successfully executed through the Build Periodically scheme, Jenkins Job will run according to the previously specified schedule. The average duration in the process of running the Jenkins Job through the Build Periodically scheme is 17.9 seconds as shown in Figure 5.

ISSN : 2615-2096 (ONLINE)
ISSN : 2615-2088 (PRINTED)

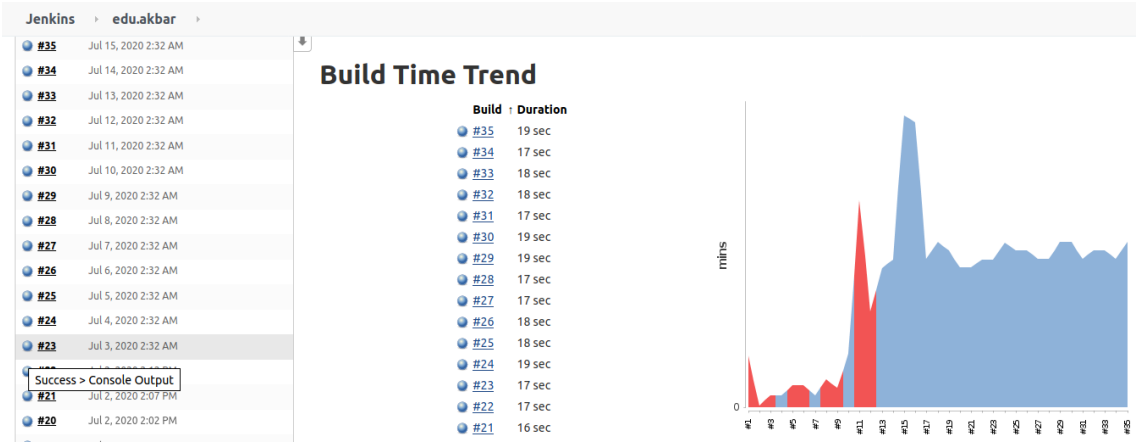

Figure 5. Test Results

CONCLUSION

Based on the results of this research, the implementation of Continuous Integration / Continuous Delivery (CI / CD) and Build: Container Platform practices on Production Server is able to help the process of integrating project documents between application developers, and can release from the results of application improvements automatically and periodically without any long delays i.e. with an average time of 17.9 seconds each release process and able to package applications on each different container within the Production Server. The development of this research is expected to implement a full set of DevOps for the application of software development methods.

REFERENCES

Admin i3, Blog Siapkah Perusahaan Anda Menerapkan DevOps, https://i-3.co.id/siapkah-perusahaan-anda-menerapkan-devops/ , 2019.

Arachchi, S. A. I. B. S., and Perera, I., Continuous Integration and Continuous Delivery Pipeline Automation for Agile Software Project Management, *MERCon 2018 – 4th International Multidisciplinary Moratuwa Engineering Research Conference*, 156-161, 2018.

Docker Documentation, Get Docker | Docker Documentation, https://docs.docker.com/get-docker/ , 2020.

Erich, F. M. A., Armit, C., and Daneva, M., A Qualitative Study of DevOps Usage in Practice, *Journal of Software: Evolution and Process,* vol. 29, no. 6, pp. 1-20, 2017.

Gitea, Gitea Documentation – Docs, https://docs.gitea.io/en-US/ , 2020.

Jenkins Developers, Jenkins User Documentation, https://www.jenkins.io/doc/ , 2018.

List, M., Using Docker Compose for the Simple Deployment of an Intergrated Drug Target Screening Platform, *Journal of Integrative Bioinformatics*, vol. 42, no.2, 2017.

Shahin, M., Ali Babar, M., and Zhu, L., Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices, *IEEE Access*, vol.5(c), pp. 3909-3943, 2017.

Threestayanti, L., InfoKomputer – DevOps, https://infokomputer.grid.id/read/121865700/devsecops-pentingnya-pendekatan-yang-mengedepankan-aspek-keamanan?page=all/ , 2019.

Ylonen, T., SSH – Secure Login Connections over the Internet, *Proceedings of the 6th USENIX Security Simposium,* pp. 37-42, 1996.